E190AR: Lab 5 Report

Charles Dawson & Peter Johnson

***Abstract*— In this lab, we develop a global path planner for our E190AR robot using a Rapidly-exploring Random Tree (RRT) with a path smoothing post-processing step to probabilistically construct a path from point to point on an arbitrary occupancy map. This report describes the algorithm used to accomplish this task, discusses the experimental setup used to test this algorithm in simulation, and provides experimental results validating our algorithm and implementation.**

I. Introduction

In our previous labs, we have implemented low-level motor control (through forward kinematics) and odometry of our E190AR robotic platform. Using those low-level motion control and proprioceptive features, we then implemented a local point-tracking ROS node that used proportional feedback control to navigate the robot towards a nearby target location. This local point-tracking function is capable of stably controlling the robot towards nearby locations, but it does not account for planning paths around any obstacles present in the robot's workspace. To effectively generate such global paths, which navigate around obstacles while guiding the robot towards the goal, we must develop a global path planning node in ROS, which will output a series of points that can be used by the local point-tracking node to guide the robot towards its destination while avoiding collisions with nearby obstacles.

The next section will describe the algorithm used to accomplish the global path-planning task, and subsequent sections will describe the setup and results of experimental verification of this algorithm in simulation.

II. Algorithm

A probabilistic road map is a discrete representation of a continuous configuration space (C-space) generated by randomly sampling the free configurations of the C-space and connecting those points into a graph. In this lab, we implement a type of probabilistic road map known as a rapidly-exploring random tree (RRT) algorithm. The algorithm for constructing an RRT is illustrated in the pseudocode below, but the basic premise is that the tree is grown outwards from the start node until an unobstructed path exists between the goal node and a node in the RRT. To grow the tree outwards, a node is randomly sampled from C-space, and a new node is added to the RRT at a random distance from the nearest node in the RRT in the direction of the random node [1], [3]. The new node is only added if it does not conflict with an obstacle in C-space. This algorithm biases growth of the tree towards unexplored regions of C-space, ensuring rapid exploration of the space (hence the name).

```
current_node = start_node
while no free path from current node to goal:
    A = Generate a random node
    B = Node in RRT nearest to random node
    C = Node a random distance from B towards A

    if no obstacle at C:
        current_node = C
```

We implement the function for checking if a collision-free path exists between two nodes by using the Bressenham algorithm to generate a line of pixels between the two node locations, then checking a provided occupancy map for collisions at each pixel. New node locations and random distances are generated from a uniform random distribution, and nearest nodes are computed by exhaustively searching the tree for the nearest node (this search has $O(n)$ time complexity, where n is the number of nodes already in the tree). We chose to exhaustively search the tree for the nearest neighbor rather than use a structure such as a kd-tree to reduce the complexity of our solution, since the trees generated in our use cases are typically not large compared to the processing capabilities of modern computers.

Once the goal node is reachable from some node in the RRT, our algorithm connects the goal node to the tree from that node and generates a path (i.e. an array) of nodes from the start to the goal by backtracking recursively through the tree. This algorithm can effectively generate paths leading from the start node to the goal node, but because those paths are randomly generated, they often meander and contain unnecessary twists and turns. To simplify these paths and reduce the number of twists and turns (allowing our robot to more effectively navigate the path), we apply the path-smoothing algorithm illustrated in the pseudocode below, based on [2].

```
for N iterations up to 5 * path length:
    select 2 random nodes from path
    if free path between 2 random nodes
        remove intermediate nodes from path
```

This path-smoothing algorithm works by repeatedly attempting to simplify the path at random, by cutting out a random portion of the path and checking if the resulting path is still collision-free. If the simplified path is still collision-free, the previous path is discarded and the new path is retained. This repeats a number of times equal to 5 times the length of the path, ensuring that a sufficient number of attempts at simplification are made.

III. Experiment Setup

For our path planning node, we used the provided PRM_Node class which has data members for its location in meters x, y, a parent node, and an array of child nodes. We also utilized the navigation message data type OccupancyGrid and the navigation messages service GetMap. The maps that we use are static and passed into the path planning node in the launch file. The launch file runs a map_server node which takes in a map .yaml file. The .yaml file specifies an image of the map, the resolution of the map, the origin, as well as the thresholds for occupancy and vacancy. The map_Service is then able to take in all of these parameters and generate an OccupancyGrid (map) for use by the rest of the path planning node.

We would test our path planning using several different maps which we would specify as an argument in our launch file. The file would run the map_server node, our prm_planning node, and rviz. From rviz we would select a point to navigate to by clicking, and then the path through the obstacles would be visualized.

IV. Experiment Results

Before implementing the path-smoothing procedure described in Section II, our algorithm was able to successfully generate paths from a start node at the origin to goal nodes selected by the user at arbitrary locations in C-space. Using RViz, we were able to both provide user input specifying the goal node and visualize the paths generated by our path-planning algorithm. Figure 1 shows some examples of paths generated without applying our path-smoothing algorithm. Note that these paths do successfully navigate around obstacles in C-space, but they are not very direct. To remove any unnecessary twists and turns, we apply our path-smoothing algorithm, yielding the examples shown in Fig. 2. These paths are noticeably smoother and more direct, showing significant improvements over the unsmoothed paths. Note that the paths shown in Fig. 1 and Fig. 2 were generated separately, and so do not correspond exactly, but these examples nevertheless demonstrate the power of our path-smoothing algorithm.

V. Conclusions

We were able to successfully implement a Rapidly-exploring Random Tree to generate robot motion plans that navigate between a start and a goal node while avoiding obstacles in the robot workspace. Additionally, we successfully implemented a post-processing step to smooth the paths generated by the RRT algorithm (which is inherently probabilistic and not guaranteed to produce smooth paths). We tested the performance of both of these systems on a variety of sample occupancy maps in simulation using RViz, finding that our global path-planning system performs well in a variety of situations. This work helps us prepare for the final competition, in which our robot will have to navigate autonomously through a maze

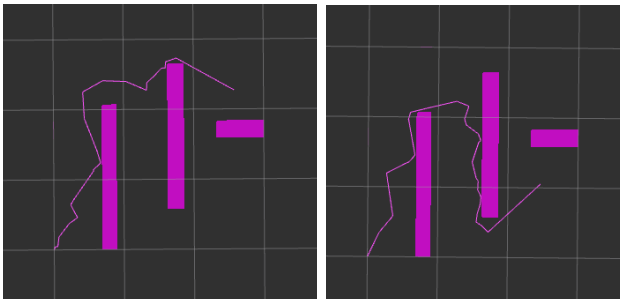

Fig. 1: Paths generated using a single-query rapidly-exploring random tree (SQ-RRT) algorithm with no path smoothing step. Note how the paths are very inefficient, containing many unnecessary twists and turns.

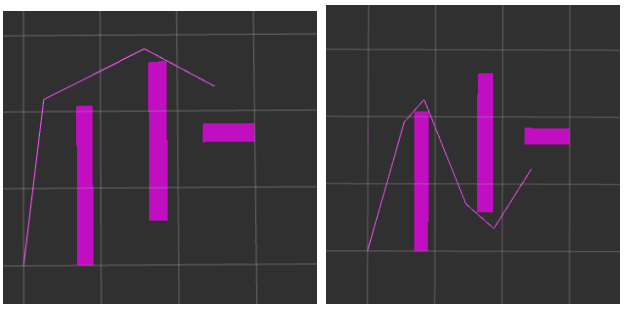

Fig. 2: Paths generated using a single-query rapidly-exploring random tree (SQ-RRT) algorithm with the path smoothing step applied. Note how the twists and turns present in the paths in Fig. 1 are not present in these paths.

References

[1] Y. Chang. E190AR. Lab Instructions, Topic: "PRM." Department of Engineering, Harvey Mudd College, Claremont, CA.

[2] D. Touretzky, E. Tira-Thompson. 15-494 Cognitive Robotics. Class Notes, Topic: "Path Planning." Carnegie Mellon, Pittsburgh, PA.

[3] âĂIJRapidly-exploring random tree,âĂİ Wikipedia, 16-Feb-2019. [Accessed: 10-Apr-2019].